Link-State Routing in Networks with Unidirectional Links

LICHUN BAO
baolc@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, CA 95064

J.J. GARCIA-LUNA-ACEVES
jj@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, California 95064
Networking and Security Center
Sun Microsystems Laboratories
Palo Alto, California 94303

Abstract—**It is shown that a unidirectional link of a network can be used for routing only if it has an inclusive cycle, which is a path that can carry routing updates from the downstream node to the upstream node joined by the unidirectional link. A new routing algorithm for networks with unidirectional links is then presented, which incrementally disseminates link-state information and selectively utilizes unidirectional links in networks. The new algorithm is verified to be correct and its complexity is analyzed. Simulations on a 20-node unidirectional network show that the new algorithm is more efficient than topology broadcasting.**

I. INTRODUCTION

Although many routing protocols and algorithms have been proposed and implemented in the past, the vast majority assume networks with bidirectional links. However, unidirectional links may occur in wireless networks because of radio link characteristics and in mixed-media networks when, for example satellite transponders are used. This paper focuses on routing in networks with unidirectional links.

McCurley and Scheider [5] presented a routing protocol for networks with unidirectional links based on complete topology information. The IETF working group on unidirectional link routing (UDLR) copes the unidirectional property of links in the network by encapsulating and tunneling IP packets in the link layer, which migrates some routing functionalities to the link layer and complicates the link layer [1]. However, UDLR assumes that a detour for a unidirectional link exists to tunnel IP packet in the reverse direction when the link is discovered. In a mobile network in which every link can change in each direction, it is hard to guarantee that the detour for a unidirectional link exist or be noticed at the time the unidirectional link is discovered. Therefore, UDLR can only solve specific cases in which the network is strongly connected and relatively stable.

Ernst and Dabbous [2] proposed a *circuit*-based link-state approach for unidirectional routing. To find out a route to a destination, a circuit including both source and destination is first detected, then validated by sending a validation message along the circuit. If a validation successfully goes through the circuit, a bidirectional communication can thus be established between source and destination, using paths on the circuit. However, when the network grows larger, the number of circuits maintained in the network becomes formidable and the algorithm has to resort to additional mechanisms in order to scale.

In this paper, we propose and verify ULP (unidirectional link state protocol), a link-state routing algorithm for networks with unidirectional links. Section II introduces the network model

and various concepts and notation used throughout this paper. Section III describes ULP. Section IV proves the correctness of ULP. Section V addresses ULP's performance, and Section VI presents the results of simulations on a network with 20 nodes used to compare ULP with topology broadcasting; the results show that ULP is a more efficient approach to supporting routing in networks with unidirectional links.

II. NETWORK MODEL

A network is modeled by a directed graph $G = (V, A)$, where V includes a set of nodes (routers) with a unique ID number, and $A \subseteq V \times V$ is the set of directed links. A bidirectional link between two nodes is represented by two unidirectional links. Link $(u, v) \in A$ if and only if v can receive information from u. We assume the link layer protocol is well designed such that information can propagate through a link with positive probability. Node u is called the *head* or *upstream node* of the link and v is called the *tail* or *downstream node*. A *cycle* in G is a directed path with distinct nodes except for the starting and ending nodes. An *inclusive cycle* for a link is a cycle that contains the link on its path.

We use the notation introduced in Table I to represent topologies, data structures and operations.

TABLE I

NOTATIONS

$u \cdot v$	Physical link $(u, v) \in A$.
$l_{u \cdot v}^i$	Link state of $u \cdot v$ reported by i.
$\partial_{u \cdot v}^i$	Inclusive cycle for $u \cdot v$ found at i.
$dp_{u \cdot v}^i$	Discovery path for $u \cdot v$ at i.
$i \Longrightarrow j$	A path from i to j.
$i \longmapsto j$	The shortest path from i to j.
$l \in i \Longrightarrow j$	Path from i to j contains link l.
$\lvert u \cdot v \rvert$	The cost of link $u \cdot v$.
$\lvert i \Longrightarrow j \rvert$	The cost of path $i \Longrightarrow j$, i.e. $\lvert i \Longrightarrow j \rvert = \sum_{u \cdot v \in i \Longrightarrow j} \lvert u \cdot v \rvert$.
TG_i	The set of link states about the network topology known by node i.
SPT_i	Shortest path routing tree of node i built on TG_i.
DG_i	Partial topology graph at node i for finding discovery path.
$TSPT_i$	Shortest path graph of node i built on DG_i.
U_i	The set of i's upstream nodes.
D_i	The set of i's downstream nodes.
$p_1 + p_2$	Concatenation of two paths p_1 and p_2.
$a \rightarrow b$	a implies b.
$a \leftarrow b$	a is assigned the value of b.

A link state $l_{u \cdot v}$ contains the following parameters in addition

Report Documentation Page		

1. REPORT DATE **1999**	2. REPORT TYPE	3. DATES COVERED **00-00-1999 to 00-00-1999**
4. TITLE AND SUBTITLE **Link-State Routing in Networks with Unidirectional Links**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **6**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

to the identifiers of the head and tail of the link:

$l_{u \cdot v}.c$ Link cost of link $u \cdot v$, $l^i_{u \cdot v}.c = |u \cdot v|$.
$l_{u \cdot v}.cs$ Inclusive cycle size;
$l_{u \cdot v}.sn$ Sequence number of the link state;
$l_{u \cdot v}.age$ The age of the link state;
$l_{u \cdot v}.rn$ The set of neighbors reporting $l_{u \cdot v}$.

For any routing protocol to work in a network, it is necessary for a two-way path to exist between any source and destination of routing information. Therefore, for a given unidirectional link $u \cdot v$, there must exist a path from v to u in order for u to receive routing information from v.

The *inclusive cycle* for link $u \cdot v$ is the shortest path from v to u that can be defined for the link, and is represented by $\partial_{u \cdot v} = u \cdot v + v \longmapsto u = u \cdot v \longmapsto u$. That is, $\partial_{u \cdot v}$ is the concatenation of link $u \cdot v$ and a path from v to u. In case of a bidirectional link, we have $\partial_{u \cdot v} = u \cdot v + v \cdot u = u \cdot v \cdot u$.

It is obvious that a downstream node should not be used as a next-hop in routing if the inclusive cycle of the link is broken, because of the resulting uncertainty regarding the link. The existence of an inclusive cycle for a link is indicated by the cycle-size property of the link, which is the summation of link costs on the inclusive cycle, i.e. $l_{u \cdot v}.cs = |\partial_{u \cdot v}|$.

To find out the inclusive cycles for links, *cycle discovery paths* for each link are maintained and propagated in the network. A cycle discovery path is the shortest path from the tail of the link to the current node, denoted by $dp^i_{u \cdot v} = v \longmapsto i$ for link $u \cdot v$ at node i. Links on discovery paths at node i compose *discovery graph* DG_i of i. Formally, $l_{u \cdot v} \in DG_i$ if it satisfies:

$$|dp^i_{u \cdot v}| < l_{u \cdot v}.cs \tag{1}$$

Suppose a link $u \cdot v$ has cycle size $l_{u \cdot v}.cs$. It is enough to let every node within a radius of $l_{u \cdot v}.cs$ from v maintain a cycle discovery path for $u \cdot v$ so that the head of the link, u, is informed of the link state eventually. The cycle size of the link is determined by algorithm (2) when the link state with its discovery path gets to the head:

$$\begin{aligned} &\textbf{if } (l_{u \cdot v}.cs > |dp^u_{u \cdot v}| + l_{u \cdot v}.c) \\ &\quad \textbf{then } l_{u \cdot v}.cs \leftarrow |dp^u_{u \cdot v}| + l_{u \cdot v}.c \, ; \end{aligned} \tag{2}$$

Fig. 1(a) shows a network with unidirectional links and Fig. 1(b) shows the discovery graph for node **n9** in that network.
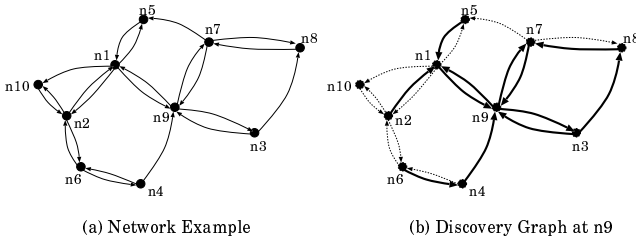

(a) Network Example   (b) Discovery Graph at n9

Fig. 1. Discovery Graph

Initially, $l_{u \cdot v}.cs$ is set to ∞ so that the link state propagates throughout the network. When its inclusive cycle is found at its head, the link state only propagates within a radius $l_{u \cdot v}.cs$ from v. A link can be used for routing by its head only if the cycle size property is set to a finite number, which implies that the head is able to know the state of that link. If the inclusive

cycle is broken and no other inclusive cycle is found for the link, the head of the link would simply reset $l.cs = \infty$ which starts another search for the inclusive cycle of the link.

III. ULP

Recent routing algorithms based on link-state information [3], [4] avoid the overhead of broadcasting complete topologies and are such that a router propagates link state updates for only those links that it uses to reach destinations. ULP adapts the link-vector algorithm (LVA) [3] to operate in networks with unidirectional links. With LVA, each router maintains its own routing tree and reports to its neighbors link state updates for those links it uses to reach destinations, as well as for those links it stops using in its routing tree.

ULP consists of three parts: *Neighbor Protocol* (NBR), *Network Routing Control Algorithm* (NET) and *Retransmission Protocol* (RET). NBR provides mechanisms for a node to detect upstream neighbors, update cycle sizes of downstream links, and propagate link states that satisfy (1). NET calculates the shortest path tree (SPT) based on Dijkstra's algorithm and sends changes in SPT to upstream neighbors. RET keeps a list of packets for retransmission upon timeout, until it receives acknowledgments from their destinations or destinations become non-neighbors.

A. Neighbor Protocol

Three data structures, D_i, U_i and DG_i, are maintained by NBR. U_i contains statistics for detecting and maintaining incoming links. D_i monitors outgoing (downstream) links and recommends these links for routing as long as they have inclusive cycles. DG_i keeps link states that satisfy Eq. (1) to find their inclusive cycles.

A.1 Link Detection

With NBR, a node periodically broadcasts a HELLO packet to inform neighbors of its existence. However, if other packets are sent out during the interval of HELLO packets, the next HELLO packet is suppressed. Upon detection of HELLO packets from a neighbor u by node i, a link $u \cdot i$ is set up and u becomes one of U_i. Without loss of generality, the cost of an active link is set to 1 ($l_{u \cdot i}.c = 1$). If the link disappears, $l_{u \cdot i}.c$ is set to ∞ by i.

A.2 Cycle Size

In ULP, both the head and tail of a link can originate link-state updates for the link. The cycle size of a link state is decided by the head of the link. The differences in link state at head and tail is resolved with the following algorithms. For a link $u \cdot v$, u detects the inclusive cycle and determines $l_{u \cdot v}.cs$, while v accepts any change on cycle size made by u for $l_{u \cdot v}$ and generates a new link state with higher sequence number.

```
// Found ∂u·v in DGu at u.          // Received l^u_{u·v}.
if ( |∂^u_{u·v}| ≠ l^u_{u·v}.cs ) {   if ( l^u_{u·v}.cs ≠ l^v_{u·v}.cs ) {
    l^u_{u·v}.cs ← |∂^u_{u·v}|;          l^v_{u·v}.cs ← l^u_{u·v}.cs;
    l^u_{u·v}.sn ← l^u_{u·v}.sn + 1;     l^v_{u·v}.sn ←
    propagate l^u_{u·v};                     max( l^u_{u·v}.sn, l^v_{u·v}.sn)+1;
}                                        propagate l^v_{u·v};
                                     }

        Procedure at u                          Procedure at v
```

A link is used for routing by its head node or an upstream node only if the link has associated with it a finite cycle size. Therefore, the decision of cycle size by the upstream node permits the node to enact a correct response when the inclusive cycle of the link is broken and the upstream node has no information about the state of the link. In section IV, we prove that the upstream node and downstream node hold *equivalent* views about the link between them (*Lemma 1*). That is, they either reliably exchange link-state updates in NBR and NET, or stop coordinating link-state information.

A.3 Neighbor States

The upstream node table U_i at node i is used to monitor incoming links. The states of an upstream neighbor are illustrated in Fig. 2.
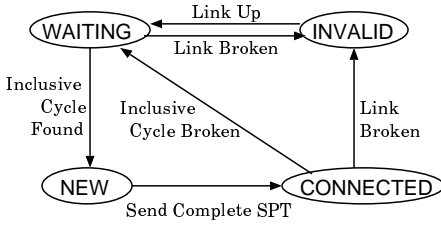

Fig. 2. State Transitions of Upstream Node

The WAITING state of an upstream node $u \in U_i$ indicates that the link $u \cdot i$ has been detected by i but is still unnoticed by u. A WAITING node becomes NEW when $\partial_{u \cdot i}$ is found by u and the new link state with finite $l_{u \cdot i}.cs$ is received by i, who sends a complete SPT_i to u. Then, the state of u switches from NEW to CONNECTED and stays in that state as far as the inclusive cycle is maintained ($l_{u \cdot i}.cs < \infty$). Updates in SPT_i are sent to an upstream node reliably only if it is in CONNECTED state. An upstream node in CONNECTED state goes back to WAITING state if the inclusive cycle is broken. An upstream node u in INVALID state means the link $u \cdot i$ disappears.
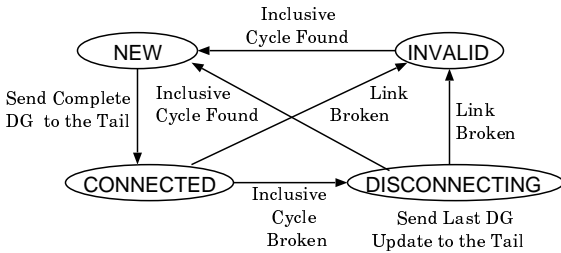

Fig. 3. State Transitions of Downstream Node

A downstream node table D_i is also maintained for outgoing links (Fig. 3). A downstream node is initialized INVALID, and enters NEW state if the inclusive cycle is found at i, when i sends a complete DG_i to that downstream node reliably. Afterwards, the downstream node becomes CONNECTED and the downstream link in CONNECTED state is given to NET for routing. If the inclusive cycle of the outgoing link is broken, the CONNECTED downstream node enters the DISCONNECTING state, until another inclusive cycle is found to change the node back to NEW state. A downstream node in INVALID state

means that the outgoing link disappears. Outgoing links in INVALID and DISCONNECTING state cannot be used for routing.

A.4 Link-State Propagation

The set of links that satisfy Eq. (1) at node i is denoted by LD_i, i.e., $LD_i = \{u \cdot v \mid |dp_{u \cdot v}^i| < l_{u \cdot v}^i.cs\}$. This set of links and other links on the discovery paths of these links compose $TSPT_i$, i.e.,

$$TSPT_i = \{l \mid (l \in LD_i) \ \lor \ \exists u \cdot v \in LD_i(l \in dp_{u \cdot v}^i)\} \quad (3)$$

The collection of discovery paths from upstream nodes is called the discovery graph, which we denote by DG_i for node i, i.e.,

$$DG_i = \bigcup_{k \in U_i} TSPT_k \quad (4)$$

DG_i is propagated to find inclusive cycles of links. Dijkstra's algorithm is run on DG_i to find the shortest paths to the tails of links and to decide whether links satisfy (1) so that updates for these links are propagated.

Changes in $TSPT_i$ are broadcast to downstream nodes and all CONNECTED downstream nodes are required to acknowledge the update; otherwise, the update packet is retransmitted by RET. Since the set of downstream nodes changes frequently in a mobile network, links that satisfy Eq. (1) and their complete discovery paths are always packed into the same packet to avoid fragmented view of discovery paths by new downstream nodes. The format of NBR packet is depicted by Fig. 4, and a link-state entry is illustrated in Fig. 5.

Transmitter ID	Packet Seq #
Dest #	Link State #
Destinations	
Link States	

Fig. 4. Packet Formats for Neighbor Protocol

Head ID	Tail ID	Age	Cost	Cycle Size	LS Seq#

Link State Entry

Dest ID	Next Hop ID

Destination Entry

Fig. 5. Entries in Neighbor Packet

B. Routing Algorithms

In ULP, the shortest path tree (SPT) of the network is computed using Dijkstra's algorithm on the topology graph. Any change in SPT is updated reliably at its CONNECTED upstream neighbors. An operation code (OpCode) is affixed to each link-state entry in the update packet, If the link state is added or updated in the routing tree, the OpCode is set UPDATE. If a link state is deleted from routing tree, the OpCode is set DELETE to indicate to upstream nodes that the link state is no longer used for routing.

B.1 Topology Graph

For the purpose of routing, NET maintains a *topological graph* (TG) and a *routing table* (R) at each node. TG_i at node i contains the SPTs of its downstream nodes and information in DG_i, i.e.,

$$TG_i = (\bigcup_{k \in D_i} SPT_k) \cup DG_i \qquad (5)$$

Node i uses DG_i to send its complete SPT_i to an upstream node in NEW state, since link states on the inclusive cycle may not be reported by i's downstream nodes. $SPT_k, \forall k \in D_i$ is used for routing to nodes unreachable by DG_i.

The shortest path routing tree SPT_i of node i is not represented separately from TG_i; it is indicated by an *onTree* tag in each link state data structure of TG_i. The routing path to a destination must use a downstream neighbor as the next-hop only if the downstream neighbor reports the remaining path to the same destination. SPT_i is represented by

$$SPT_i = \{l \mid l = u \cdot v$$
$$\wedge \, \exists k \in D_i (i \longmapsto v = i \cdot k \longmapsto u \cdot v$$
$$\wedge \, \forall l' \in k \longmapsto v, l' \in SPT_k \cup DG_i) \}$$

Transmitter ID	Source ID
Packet Seq #	
Dest #	Link State #
Destinations	
Link States	

Fig. 6. Packet Formats for Routing Packet

Whenever SPT_i changes, a routing packet containing the routing changes with operation codes is sent to all upstream neighbors of i in CONNECTED state. The routing packet format is presented in Fig. 6. Since U_i may contain several upstream nodes and an update packet takes several hops to reach these upstream nodes, a single update packet is constructed with destination and next-hop information. Only those downstream nodes whose address is one of the next-hop addresses in the packet process or forward the packet. The packet later splits into several packets on the way to upstream nodes because paths to these upstream nodes may diverge at some points. This mechanism avoids that every downstream node tries to forward the packet to upstream nodes and multiple copies be generated.

The routing table R_i at node i is a vector of entries containing following information derived from SPT_i: destination dst, distance d to dst and next-hop address n.

B.2 Information Processing

Most link-state algorithms use sequence numbers to decide which link state updates are up to date. In ULP, we assume a simple sequence number scheme for updating link states. A link state l is considered more recent than the one in TG_i if $(l \notin TG_i \vee l.sn > TG_i.l.sn)$. A new link state is updated in TG_i according to the three cases discussed below.

B.2.a Link Cost Changes:. The cost of a link is monitored by the tail of the link in NBR. Link cost changes are propagated in NBR reliably because the upstream link is part of $TSPT$ according to Eq. (1) and changes in $TSPT$ are propagated to CONNECTED downstream nodes reliably. The new link state keeps propagating as long as the link satisfies Eq. (1), and eventually the head of the link is informed of the new link state by NBR. Because DG is part of TG by Eq. (5), TG also receives the new link state, which causes another calculation of SPT using Dijkstra's algorithm. Modifications to SPT at a node are sent reliably to all CONNECTED upstream routers of that node, and every node using $l_{u \cdot i}$ is notified of the most recent link state.

B.2.b Cycle Size:. The cycle size of a link is monitored by its head in NBR. Modifications to DG result in re-inspection of every downstream link to see if its cycle size changes. Consider link $i \cdot k$, the discovery path $dp^i_{i \cdot k} = k \longmapsto i$ and link cost $l^i_{i \cdot k}.c$ in DG_i are combined to decide the cycle size $l^i_{i \cdot k}.cs$. If $|k \longmapsto i| + |i \cdot k| < \infty$ and is different from the old one, $l_{i \cdot k}.cs$ is assigned the new inclusive cycle size and propagated to downstream nodes. If the inclusive cycle for link $l^i_{i \cdot k}$ is broken, i.e., $|k \longmapsto i| + |i \cdot k| = \infty$ $l_{i \cdot k}.cs \leftarrow \infty$ and another search for inclusive cycle begins in the network.

B.2.c Link Deletion:. A downstream node indicates the deletion of a link state by setting the cost of the link to ∞. The new link state is propagated by NBR to all CONNECTED downstream nodes reliably, and gets to the head of the link eventually. Because no updates to a deleted link will be generated thereafter, the link state is erased finally from all topology graphs by aging. The deletion of a link state due to aging is not propagated to neighboring nodes. When a downstream node is deleted, all link states reported by that node should be deleted. The deletion of a link state reported by a specific node simply deletes that node from reporting the node set of the link. A link l is completely erased from TG_i when its reporting node set is empty. If the deletion of a link causes SPT to change, those changes are reported to upstream nodes in CONNECTED state.

C. Retransmission Protocol

Update packets of both NET and NBR require reliable transmission to CONNECTED upstream nodes and CONNECTED downstream nodes, respectively. The retransmission protocol (RET) provides the mechanism to keep these two types of packets and retransmit them to neighboring nodes if they are not acknowledged upon time-outs. The time-out value is set proportional to the cycle size of the link between neighbors in CONNECTED state.

Given that link states keep changing in TG_i and DG_i, packets in RET only specify link-state identifiers, which are *(head, tail)* pairs. Destination entries of packets are also saved in the RET packet list, as long as those destinations are CONNECTED. If a neighbor turns into a state other than CONNECTED, that neighbor is deleted from destination entries of packets in the retransmission list. If a packet waiting for retransmission has no destination, it is removed from retransmission queue.

IV. Correctness of ULP

Lemma 1: The knowledge about the link at its head and tail is equivalent.

Proof: Consider two nodes, u and v; there are four possible cases to consider, based on the link cost $l_{u \cdot v}.c$ and cycle size $l_{u \cdot v}.cs$ of a link state $l_{u \cdot v}$:

1. $l_{u \cdot v}.c < \infty \ \wedge \ \exists \partial_{u \cdot v} \in G$, which means an inclusive cycle exists for the link. Changes to $l_{u \cdot v}$ at both u and v are able to get to the other one by NBR, since v accepts $l_{u \cdot v}^u.cs$ regardless $l_{u \cdot v}^u.sn$ and always increases $l_{u \cdot v}.sn$. Because u updates $l_{u \cdot v}$ only if there is a difference between $l_{u \cdot v}.cs$ and the actual inclusive cycle, u and v get consistent $l_{u \cdot v}$ in this case.

2. $l_{u \cdot v}.c < \infty \ \wedge \ \partial_{u \cdot v} \notin G$, which means the link exists but the inclusive cycle is broken. Since the broken link propagates down the path on inclusive cycle and gets to u, u resets the cycle size and propagates the new link state to v. v becomes DISCONNECTING downstream node of u and u is a WAITING upstream neighbor of v. RET at u keeps sending the new link state to v for limited times to insure that v get the new link state because acknowledgment from v is unable to reach u. u and v stop coordinating link-state updates after a finite time.

3. $l_{u \cdot v}.c = \infty \ \wedge \ \exists \partial_{u \cdot v} \in G$, which means the link is broken. In this case, the new link state is propagated down the inclusive cycle in NBR, and u gets to know the link state within finite time after the link disappearance. Both u and v are in INVALID state at each other.

4. $l_{u \cdot v}.c = \infty \ \wedge \ \partial_{u \cdot v} \notin G$, which means the link is broken and the inclusive cycle also breaks before the new link state propagates to u. In this case, since u knows the exact information about $l_{u \cdot v}.cs$, v is in DISCONNECTING state at u and u is in INVALID state at v. u and v do not coordinate link-state updates. $\square$

Lemma 2: An upstream node reliably gets the shortest path tree of its downstream nodes in CONNECTED state.

Proof: In NBR, an upstream node is CONNECTED at the downstream node if the link has a finite inclusive cycle, which implies that it is a downstream node at its upstream node. Since both upstream node and downstream node hold equivalent link state about the link between them (*Lemma 1*), they are both in CONNECTED state at each other. Thus, link-state updates in the SPT of the downstream node are transmitted to the upstream node. By RET, the update packet keeps being transmitted, until it is acknowledged by the upstream nodes. Therefore, the upstream node reliably receives updates of SPT from its downstream nodes in CONNECTED state. $\square$

Theorem 3: ULP stabilizes within a finite time when the network topology stops changing.

Proof: After any sequence of topology changes in the network, both the head and tail of a link keep equivalent link state by *Lemma 1*, and every upstream node receives link-state updates from its CONNECTED downstream nodes by *Lemma 2*. Since a node stops propagating a link state once it already has up-to-date link-state information, the propagation of an up-to-date link state happens only once at each node. By our finite model of a network, it is finite time to stabilize coordination of link-state updates between neighboring nodes. That is, the time for stabilizing the distributed algorithm is finite, and ULP stabilizes within a finite time. $\square$

Theorem 4: When ULP stabilizes, there is no loop in network routing.

Proof: We prove the claim by contradiction. Assume that there exists an implicit loop at i in routing to j. Since i runs Dijkstra's algorithm based on local topology information, $i \longmapsto j \in SPT_i$ contains no loop. Assume $i \longmapsto j = i \cdot k \longmapsto j, k \in D_i, k \longmapsto j \in SPT_k$. Since $i \notin i \longmapsto j, i \notin k \longmapsto j$.

As assumed, i stays on a routing loop, $\exists p \cdot q \in k \longmapsto j$, where $i \notin p \longmapsto j$ but $i \in q \longmapsto j$. Because the algorithm stabilizes, we know that $p \longmapsto j = p \cdot q \longmapsto j$ by its definition, which implies $i \in p \longmapsto j$. This contradicts with $i \notin p \longmapsto j$. $\square$

V. Performance Evaluation

The communication complexity of NBR can be analyzed probabilistically. Assume the probability of link-state changes at any time for every link is p, the communication complexity of NBR within a time unit in terms of number of link states propagated is:

$$C_{nbr} = p \cdot \sum_{i \in V} \sum_{\substack{u \cdot v \in A \\ \wedge |dp_{u \cdot v}^i| < l_{u \cdot v}.cs}} |dp_{u \cdot v}^i| \qquad (6)$$

For example, suppose every link has link cost of 1 and inclusive cycle size of 3, every node has in-degree of 2 and every two nodes within two hops are distinct, then the link state is propagated within limited distances by NBR, and the communication complexity is $p \cdot (1 \times 2^1 + 2 \times 2^2 + 3 \times 2^3) \cdot |A| = 34 \cdot p \cdot |A|$ for the network, because new link states with their discovery paths are propagated within two hops.

The communication complexity of NET is difficult to estimate when the network routing is already established and link states change randomly. However, the overall communication complexity of the routing algorithm at network start-up can be estimated by Eq. (7), because the downstream node of a link sends its complete SPT through path on the inclusive cycle to the upstream node when the inclusive cycle of the link is found at the upstream node, and each routing tree SPT contains V entries.

$$C_{net} = |V| \cdot \sum_{l \in A \wedge l.cs < \infty} (l.cs - l.c) \qquad (7)$$

To compare the efficiency of data communication in the network, we introduce the concept of *routing weight* to compare the data traffic cost that results from different routing algorithms. The routing weight is the summation of finite distances to other nodes at every node in the network. It provides a comparable metric for efficiency of bidirectional and unidirectional routing algorithms when the network has the same connectivity from both bidirectional and unidirectional points of view. Two networks $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ have *the same connectivity* if $\exists f : V_1 \rightarrow V_2$ such that $\forall u, v \in V_1$, $f(v)$ is reachable from $f(u)$ in G_2 if and only if v is reachable from u in G_1. A higher routing weight indicates that longer paths are used, which is less efficient. By enabling the use of unidirectional links into a network routing protocol, we can shorten the distance for data traffic from source to destinations. If we assume each node generates equal traffic to every other nodes with uniform rate K, the cost of network data communication within a

time unit would be:

$$C_{data} = K \cdot \sum_{i,j \in V} R_i^j . d \qquad (8)$$

which is proportional to the routing weight of the network.

Fig. 7 is a sample network with unidirectional links. Within the network, a spanning tree connects all nodes with bidirectional links so that LVA can work correctly in this network. If any link on this bidirectional links breaks, LVA is not comparable with ULP. Therefore, we only illustrate the advantage of ULP over LVA by calculating routing weights when no link breaks. Assuming every link has a unit cost, the routing weight of LVA is 1898, while the routing weight of ULP is 1530; hence, efficiency is improved 20% over LVA.

In general, assuming the same number of links in the network, unidirectional link decreases network load by shrinking the distance from one node to the other according to Eq. (8). However, the communication overhead on NBR and NET in unidirectional network is more than that of bidirectional network due to long-haul neighboring relationship as implied by Eqs. (6) and (7).

VI. SIMULATIONS

Obviously, ULP consumes more network resources maintaining long-haul neighbor relationships than transitional routing algorithms that operate over bidirectional links only, where a node's neighborhood lies within only one-hop distance.

We simulated an ideal link-state algorithm based on OSPF, called EOSPF (extended OSPF), whose neighbor protocol incorporates unidirectional links like ULP does. The difference between ULP and EOSPF is that EOSPF uses flooding to propagate link-state updates to all CONNECTED upstream nodes, while ULP selectively send link-state updates to CONNECTED upstream nodes if the link-state updates result in different routing paths.
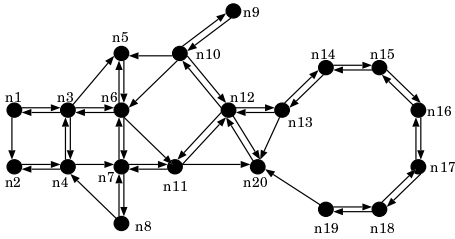

Fig. 7. A 20-Node Sample Network Topology

Fig. 8 shows simulation results of ULP and EOSPF on the 20-node network of Fig. 7. The effects of five kinds of single topology changes were measured, including link additions, link deletions, link cost increments by 1, node additions and node deletions. The first column of Fig. 8 records the accumulated number of new link states at all nodes after each single topology change in the network; the second column reflects the accumulated number of routing update packets after each single change. EOSPF consumed much more network resource than ULP did as indicated by the second column. ULP consumes less network resources at the expense of longer convergence times than EOSPF.
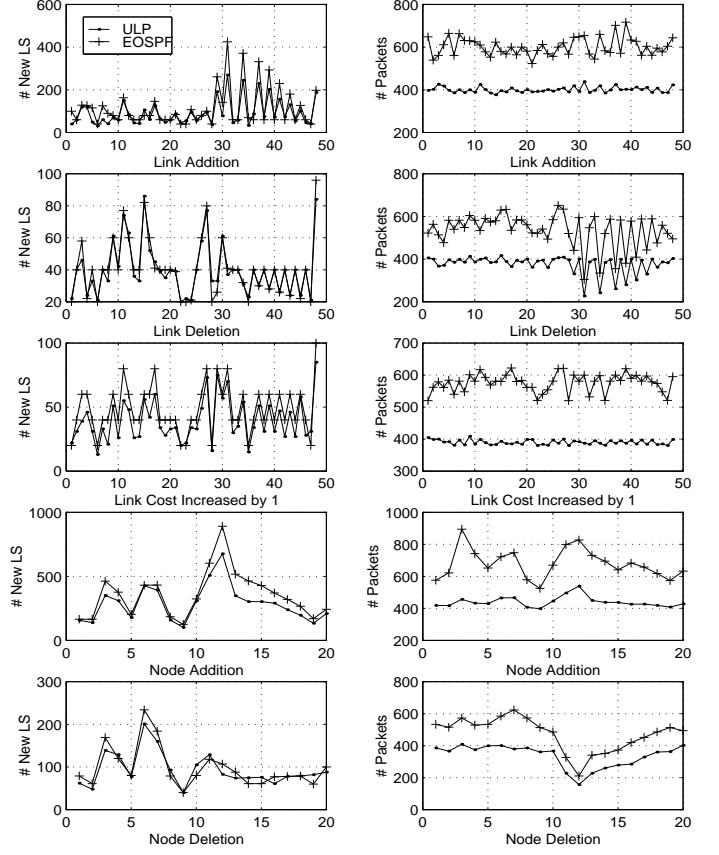

Fig. 8. Simulation Results on a 20-Node Unidirectional Network

VII. CONCLUSION

Our contribution in this paper is the introduction of the inclusive-cycle size property of a link state, which is an important criterion in routing and link-state maintenance. We also defined the concept of routing weight as a measure for evaluating the efficiency gained with routing algorithms that can incorporate unidirectional links over similar algorithms that require bidirectional links.

Although ULP requires less overhead than EOSPF in networks with unidirectional links, there is much to improve over ULP. One drawback of ULP is that the searching process of inclusive cycle consumes a lot of network resources by initializing the cycle size of a link to ∞ and flooding the link state throughout the network. This can be improved by limiting the distance of the link-state propagation in search for the inclusive cycle.

REFERENCES

[1] Walid Dabbous, Yongguang Zhang, David Oran, and Rob Coltun. http://www.ietf.org/html.charters/udlr-charter. html.
[2] Thierry Ernst and Walid Dabbous. A circuit-based approach for routing in unidirectional links networks. Technical report, Inria, Institut National de Recherche en Informatique et en Automatique, 1997.
[3] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, Oct. 1995.
[4] J.J. Garcia-Luna-Aceves and M. Spohn. Scalable link-state internet routing. In *Proc. IEEE International Conference on Network Protocols (ICNP 98)*. Austin, Texas, Oct. 14–16 1998.
[5] Robert McCurley and Fred B. Schneider. Derivation of a distributed algorithm for finding paths in directed networks. *Science of Computer Programming*, 6(1):1–9, 1986.